LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

LLNL-TR-654552

Review of Graph Databases for Big Data Dynamic Entity Scoring

M. X. Labute, M. J. Dombroski

May 16, 2014

Disclaimer

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Table of Contents

Summary

Modeling data as a graph enables users to quickly analyze networked phenomenon, such as social network-based marketing data (e.g., linking entities in social media based on their friends and their "likes", their friend-of-a-friend's "likes", etc.) and scientific data, such as in biology to assess a gene's proximity in a graph to particular disease phenotype. Numerous purpose-built graph databases (DBs) called “NoSQL” or “Not Only SQL” DBs are emerging to support this type of graph analysis. These DBs differ from traditional relational DBs in that they are optimized to improve storage and query performance across large, complex graphs.

This document reviews the state of the art in graph DBs and identifies potential DBs that show promise for dynamic entity-scoring algorithms. Dynamic entity scoring is a new area of research that allows analysis of data within graphs using undirected methods to weigh the edges of the graph based on the proximity or ambient influence of near-by nodes. The EntityScore (eScore) methodology developed at Lawrence Livermore National Laboratory (LLNL) is one application of these emerging graph-analysis algorithms to help analysts identify and prioritize entities, based on complex, analyst-defined weighting criteria.

Our review examines a wide range of different, well-known graph DBs in use since at least 2010 and identifies unique advantages and disadvantages that may impact eScore-specific requirements. Specifically, we review data structures employed, query features employed, load times, query calculation times, memory usage and several other important attributes reported in the literature. We limited our considerations for eScore to Blueprints-compliant graph DBs (https://github.com/tinkerpop/blueprints/) because they provide a standard, java application programming interface (API) based on the property graph model.

Some major findings we uncovered include:

- Performance varies greatly across different DBs depending upon the size of the graph and how well-optimized a given DB is for a particular task
- There is not a one-sized fits all solution; several different graph DBs provide a range of useful features for different applications
- For the eScore application, it appears that open-source, Blueprints-compliant DBs such as DEX and Titan may be best suited

Graph Database Background

Query performance and data modeling limitations presented by traditional relational models underlying most enterprise DB systems (e.g. RDBMS) in emerging application domains has led to a movement away from tabular, relational DBs towards more intuitive data storage and persistence technologies (known as the ‘NoSQL’ movement).[1] Relational DBs are plagued by

performance and data modeling issues such as lengthy load times for some problems and limited functionality in modeling and querying data. Four prevalent DB models have emerged through the 'NoSQL' movement promising to address performance and data modeling issues: (1) Wide-column stores, modeled after Google's Bigtable (e.g., Cassandra), (2) Document stores (e.g., MongoDB), (3) Key-value stores (e.g., Accumulo, BerkeleyDB), and (4) graph DBs. **Figure 1** illustrates the tradeoff in data complexity vs. data size across these four technologies; graph DBs fit for domain applications where the complexity of the data is more of a factor than sheer size.[2]

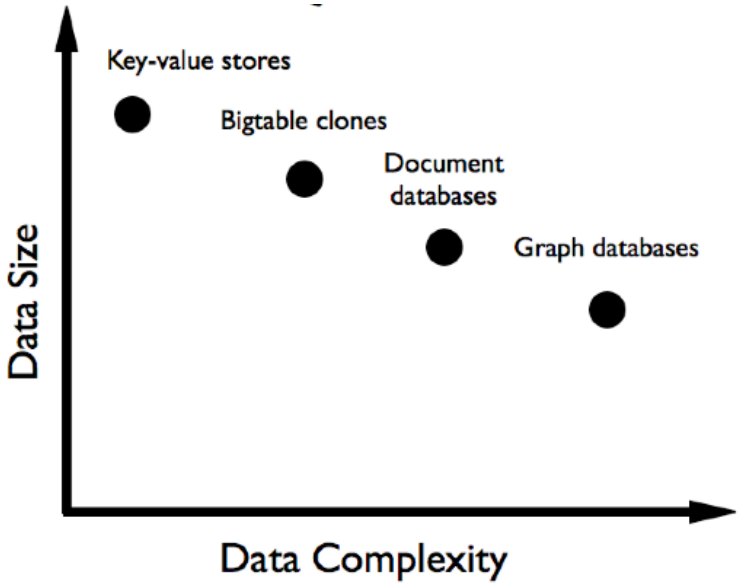

Figure 1. Different "NoSQL" data models.[2]

More specifically, graph DBs are well-suited for situations in which the source of the data complexity is rooted in the relationships between the entities rather than the "intrinsic" data associated with a single entity. Graph DBs are particularly useful in situations where the data is best represented as a graph or network of nodes, edges, and properties that a user can assign to nodes and edges. Examples of where they might be useful include social network analyses where understanding complex relationships between comingled entities is important.

The recent massive growth in Web 2.0 social network graphs, graph analytic capabilities, and distributed computing architectures has led to a resurgence in of interest in graph DBs. Graph DBs was an active research area from the early-to-mid 90s and then greatly diminished as other data models such as XML became widely used. However, social network graphs belonging to various Internet companies and the inherent interconnectedness of the Web itself spurred interest in graph analytic research throughout the early 2000s. Additionally, network visualization of chemical and biological processes has made graph analytics in general and Semantic Web-related applications in particular very popular in the life sciences recently. Over the past 5–10 years, distributed computing technology platforms (e.g. Hadoop ecosystem, AMPLab stack) have appeared that can handle analytics at massive scale. Likewise, distributed graph processing software has been developed, sometimes operating on top of these stacks, or as stand-alone technologies.

Due to the explosion in graph DB applications, there has been a development in the 'NoSQL' movement towards Blueprints-compliant DBs because it provides a generic, java API that is based on the property graph model, which appears to be sufficient for a wide range of graph-related problems. The property graph model is a directed, edge-labeled, attributed multigraph consisting of nodes and edges that can have labels and values for node/edge attributes. Blueprints defines a basic interface for property graphs that can be used to interact with a graph. Basic operations include: adding/removing vertices and edges, retrieving vertices/edges by an identifier (ID), and retrieving vertices and edges by an attribute value. An application that is built on a Blueprints graph DB can be easily switched to another Blueprints-graph DB, so applications can be developed in a way that is agnostic to the choice of DB. In addition, Blueprints graph DBs are compatible with tools such as Rexter, which is a graph server so one can remotely access a graph DB, and Gremlin, a query language for doing graph traversals. However, a key drawback of Blueprints is there is a lot "buried under the hood" of the API and some features cannot be configured, such as the details of data updating. Despite this limitation, the remainder of this review focuses primarily on DBs that meet the Blueprints-compliant standard. Given the recent resurgence in graph DB development and the scope of this assessment, a key question is: what are the relevant metrics to compare graph DBs and choose one for an application domain?

Graph Database Requirements for eScore Applications

The eScore suite of algorithms requires the ability to extract and transform data, which will then be loaded into a weighted node- and edge-attributed property graph. The algorithms also require running multi-hop path traversals and the ability to match complex subgraph query patterns of interest in interactive times over graphs of various sizes. We expect that the data will take the form of a single batch upload of historical records, with subsequent periodic (daily- or weekly-) updating of the graph.

Using this implementation, we can identify some basic requirements that would be helpful in a graph DB. Although an open source code base is not a requirement, it is a feature that we favor. The eScore approach will require several important graph DB features including:

- Data model flexibility (i.e., capability to iteratively modify or swap out ontologies to accommodate new applications or evolving domain application requirements)
- Generic data store, offering flexibility in switching between different data stores
- Capable of supporting a broad range of queries across the graph (e.g., node/edge adjacency, simple paths and shortest path, pattern matching)
- Performance scales predictably with graph size
- Ability to integrate with 3rd party software

We are uncertain how eScore will perform on different platforms and what specific requirements are necessary for a specific eScore application. Therefore, we will implement eScore to the Blueprints standard and for the time being defer deeper questions regarding specific application requirements and features until the algorithm is applied to specific use-cases. This way we could swap out different Blueprints graph DBs optimized to the application space.

The literature reports two different approaches for comparing graph DBs: one that compares different graph DBs based on a carefully chosen set of implementation-independent qualitative features; the other a more quantitative, implementation-dependent benchmark-oriented performance assessment. Together both of these assessments could build a comprehensive assessment of different DB options. We review the literature here for both qualitative and quantitative comparisons with specific reference to the eScore requirements laid out above.

Qualitative Comparisons of Graph Databases

A recent important qualitative comparison was performed by Angles (2012),[3] who compared current graph DBs on the basis of their underlying DB model. They identified three components common to all graph DB models: (1) the schema and data instances modeled as graphs, (2) graph-oriented operations and type constructors (i.e. defining node type), and (3) a set of integrity rules (i.e. related to data consistency under various operations). They advocated using the DB model as a set of features to compare one graph DB to another since comparing features ensures innate support needed for querying and storing the data of interest to an application.

Angles compared a range of different graph DBs that may be useful to an eScore application. Proprietary DBs included in the Angles assessment include Allegrograph and InfiniteGraph. All other graph DBs are open source, although Neo4j has an advanced proprietary version of its DB that can be purchased. Although Angles did not assess Titan in his assessment of graph DBs, we made assessments of its features and added Titan to the assessment.

We recapitulate the information from Angles (2012)[4] here, and focus on those features most important to the eScore requirements. **Table 1** lists features related to data storage and manipulation as well as the data structures needed to represent a graph. **Table 2** compares features related to query language properties and checks that are done to ensure data integrity as new data is added to the existing graph.

Table 1. A comparison of data-related features.

Feature	AllegroGraph	DEX	InfiniteGraph	Neo4j	Titan
Main memory storage	X	X		X	X
Indexes*	X	X	X	X	X
GUI	X				
Simple graphs†	X				
Attributed graphs‡		X	X	X	X
Node labeled	X	X	X	X	X
Node attribution		X	X	X	X
Directed Edges	X	X	X	X	X
Edge labeled	X	X	X	X	X
Edge attribution		X	X	X	X

Table 2. A comparison of query and data integrity features across the graph DBs.

Feature	AllegroGraph	DEX	InfiniteGraph	Neo4j	Titan
Query Language§	X			X	X
Query API	X	X	X	X	X
Blueprints-compliant		X	X (read only)	X	X
Retrieval Queries**	X	X	X	X	X
Reasoning Queries††	X				
Graph Analytics Queries‡‡	X	X			
Types checking		X	X		

Table 3 shows the level of support offered by the different graph DBs for essential queries. Essential queries range from simple tests of whether two nodes are adjacent through an edge, to finding the shortest path between two nodes, to matching complex graph patterns.

* Indexing data inputted into a graph is essential for data retrieval operations.

† A collection of nodes and edges where edges are undirected, no loops exist, and there is no more than one edge between two different nodes.

‡ Users can specify user-defined characteristics to describe node and/or edge properties.

§ A collection of operators or inference rules used to retrieve and manipulate data in the DB.

** Queries focused on retrieving nodes or edges and their characteristics.

†† Queries focused on pattern matching or more sophisticated processing operations than simple retrieval.

‡‡ Queries supported by a dedicated graph analytics (pattern matching, identifying unknown relationships, edge/node inference, etc.) engine incorporated into the DB.

Table 3. Support for "essential" queries across the graph DBs.

Feature	AllegroGraph	DEX	InfiniteGraph	Neo4j	Titan
Node/edge adjacency	X	X	X	X	X
k-neighborhood*					X
Fixed-length paths†	X	X	X	X	X
Regular simple paths‡		X	X	X	X
Shortest path§		X	X	X	X
Pattern matching	X	X	X	X	X

Based on this qualitative assessment, DEX, Neo4j, and Titan provide distinct data feature advantages useful to eScore with the capability for main memory storage and attributed graphs. Essential queries are best supported by Neo4j, Titan, DEX, and InfiniteGraph. Although Allegrograph provides some useful data features and reasoning queries (which other DBs do not support), Allegrograph may not be useful to the eScore application due to limited Blueprints-compliance and an inability to support several essential queries.

Quantitative Comparisons of Graph Databases

Testing graph DBs against one or more benchmark metrics allow a quantitative comparison of different DBs. DB loading and query performance often varies as a function of dataset size. There are two major issues with doing a quantitative comparison. The first is that it is difficult to compare equitably, given the differences in optimizations that have been done to a given graph DB. For example, one DB might be optimized for queries that feature cross products and/or complex joins, and therefore perform well on these queries. High performance on complex queries could lead one to incorrectly infer similar efficiency with simpler queries, while in reality a DB might not perform as well as other DBs that can't handle the complex queries. The second issue is that because of the huge differences in datasets, it is non-trivial to evaluate the relevance of a benchmark to one's set of data and queries; therefore, one must interpret benchmarking results with caution.

There have been four major benchmarking studies on current graph DBs. The first by Jouili et al. compares Neo4j, OrientDB**, Titan, and DEX on a 250,000 node (1,250,000 edge) graph and a 500,000 node (2,500,000 edge) graph.[5] The graphs were synthesized by a Barbasi-Albert model and had power-law degree distributions with a mean node degree set to 5. Only a single thread was utilized to load the DB. The authors compared load times, graph traversal workloads (i.e., shortest path and breadth first search exploration of the nearest neighbor nodes around a given node), and intensive workloads (i.e., obtaining a vertex by querying for its ID, obtaining a vertex

* A non-parametric algorithm for identifying class membership among *k* nodes or identifying an average property value regressed across *k* nearest neighbors.

† An algorithm used to identify a route between two nodes using a specified number of hops.

‡ An algorithm used to identify a path between two nodes that satisfies a regular expression (e.g., a specific set of properties on the traversed edges).

§ An algorithm used to identify a path between two nodes that minimizes the sum of traversed edge weights.

** OrientDB is an open-source, Blueprints compliant DB.

by one of its properties, and adding edges). The graph DBs were run on a 2.5 GHz dual core machine with 8 GB of physical memory. Representative selections of results are discussed below. In **Figure 2** we show load time results for all DBs except OrientDB since it took 588s s to load the 1000k element graph in OrientDB.

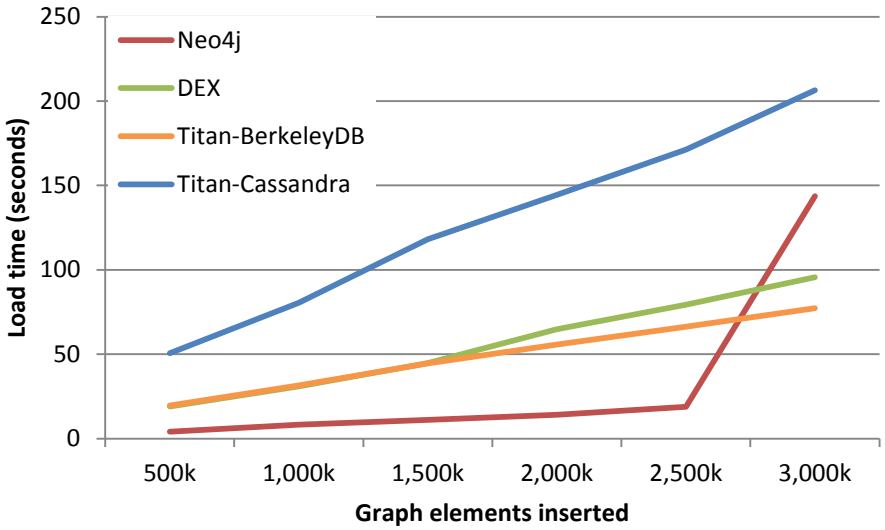

Figure 2. Load times as a function of graph size for 4 different graph DBs from Jouili et al.

In addition to loading data into the graph DBs, Jouili et al. also investigated two types of canonical graph analytics workloads: (1) Shortest path where the task is to find all paths that include less than a certain number of hops between two randomly chosen vertices and (2) neighborhood breadth-first search exploration where the task is to find all vertices that are a certain number of hops away from a randomly chosen vertex. The calculations were done on the 500k graph and they performed 100 measures of each workload, selecting different starting vertices each time. The interquartile ranges of return times for the shortest path workload were well below 1.5 s for all DBs for 1 hop paths. For 2 and 3 hops, the median times increase linearly but the interquartile ranges grow faster, exhibiting large variances. Neo4j is systematically the best at this task and Titan-Cassandra is the slowest. For neighborhood breadth first search, the results are similar. As we go from1 to 3 hops away from the chosen vertex, we stay under 1 second across all the DBs, but Neo4j is systematically the best and Titan-Cassandra is the slowest and has the largest variance across return times.

An additional task was investigated where an edge was added to two randomly selected nodes in the graph. DEX and Titan-Cassandra were the clear winners at this task and the times were reduced as the number of clients increased. Neo4j was not as effective at this task (the times were 4x or greater than other DBs) and the results seemed insensitive to the number of clients added.

Based on this analysis, it appears that graph DB performance depends upon the task that is being done. Neo4j was the fastest at calculating shortest path and neighborhood-breadth first search. However, DEX and Titan-Cassandra were the fastest when adding a new edge between two

randomly selected nodes. Figure 2 shows that DEX and Titan have predictable load times for different sized graphs, whereas Neo4j loads quickly for small graphs, but is much slower when 3,000k elements are added. For the eScore application these results suggest that DEX, Titan, and Neo4j offer potential advantages.

The second study by Ciglan et al. benchmarks multi-hop traversal operations, comparing SGDB*, Neo4j, Sail†, OrientDB and DEX using data sets of 1,000, 10,000, 40,000, 50,000, and 100,000 vertices with a mean vertex degree of 16 for each of these and some larger datasets, culminating in a 1 million node graph.[6] The authors collected results for the load times for the smaller datasets and two traversal operations: (1) computation of breadth-first search 3 hops from a vertex for 10,000 randomly chosen vertices, and (2) computation of connected components. The benchmarks were designed to test graph DB capabilities in a memory constrained environment, so the experiments were run on a low-end machine with a 2-core 2.4 GHz Intel processor and only 2 GB of RAM. The max heap size for the Java virtual machine (JVM) was set to 1.5 GB. The values reported were averaged over ten runs of that benchmark.

Their results are shown below in **Figures 3 through 5**.

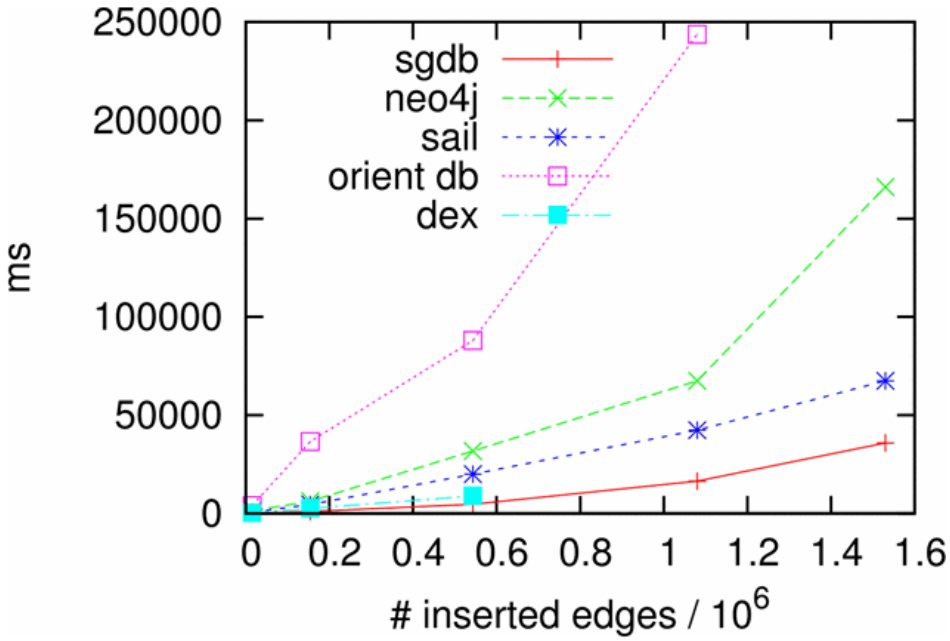

Figure 3. Load times for graph datasets of various sizes.

* SGDB is a research-based, Blueprints compliant prototype DB.

† An open-source DB that is somewhat compliant with Blueprints under certain conditions (https://github.com/tinkerpop/blueprints/wiki/Sail-Implementation).

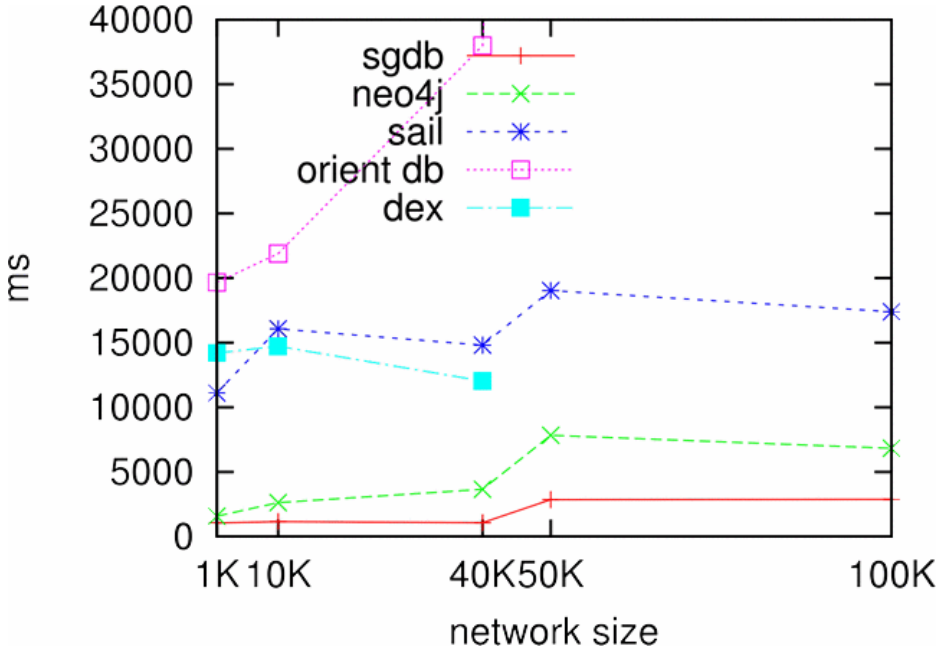

Figure 4. Computation of breadth-first search explorations of the neighborhood 3 hops away from a vertex for 10,000 randomly chosen vertices.

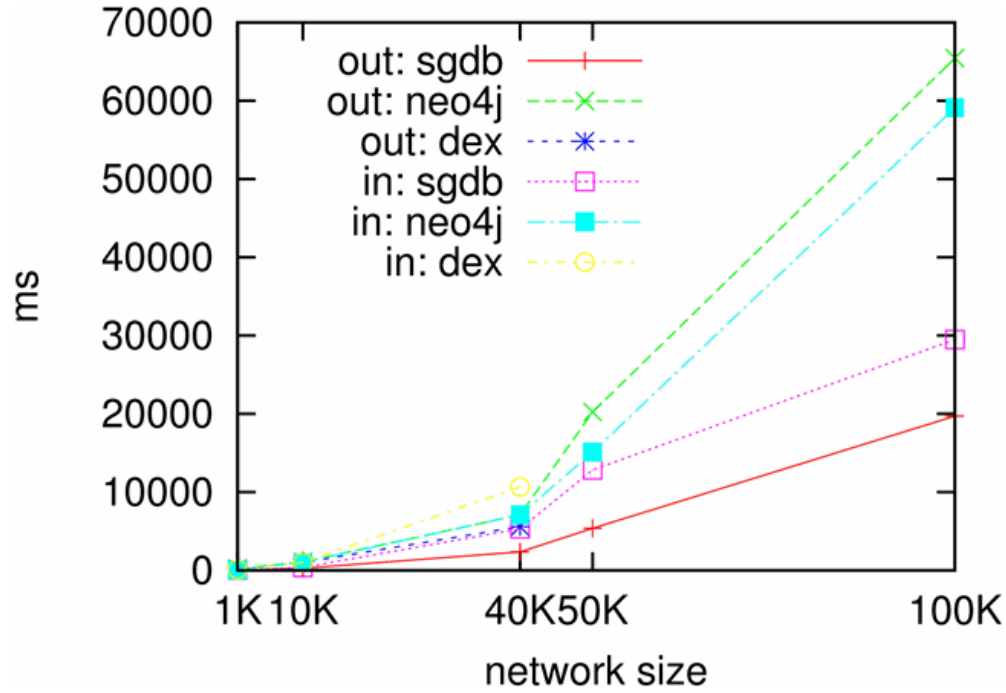

Figure 5. Computation of connected components using in- and out- going edges.

These results indicate that Sail, DEX, and SGDB have predictable load times over large graphs. Neo4j and SGDB perform breadth-first search explorations fastest. SGDB performance scales predictably over network size for computing connected components. For the eScore application, SGDB may warrant further consideration based on these results. Neo4j, DEX, and Sail also show potential utility.

The third study by Dominguez-Sal evaluates different graph DBs according to graph analysis benchmarks used in high-performance computing (HPC) graph processing.[7] The authors compared Neo4j, HypergraphDB*, Jena†, and DEX on graphs generated using the R-MAT algorithm, which is able to build graphs of any particular size and edge density. R-MAT graphs typically follow power law degree distributions, which correspond well to real-world graphs (e.g., biological systems, social networks). The generated data consisted of 1,000 node, 32,000 node, and 1M node graphs. The benchmarking consisted of performance assessment on four different operations: (K1) time for edge/node insertion, (K2) time needed to find all the edges with the largest weight, (K3) time spent to build subgraphs in the neighborhood of a node which relies on output from (K2) and finally (K4) traversed edges per second while calculating the betweeness centrality for each node in the graph. The experiments were executed on a computer with two quad core Intel Xeon E5410 2.33 GHz processors with 11 GB of RAM and a 2.25 TB Large Form Factor disk drive. **Figure 6** and **Figure 7** show K1 and K4 results, respectively.

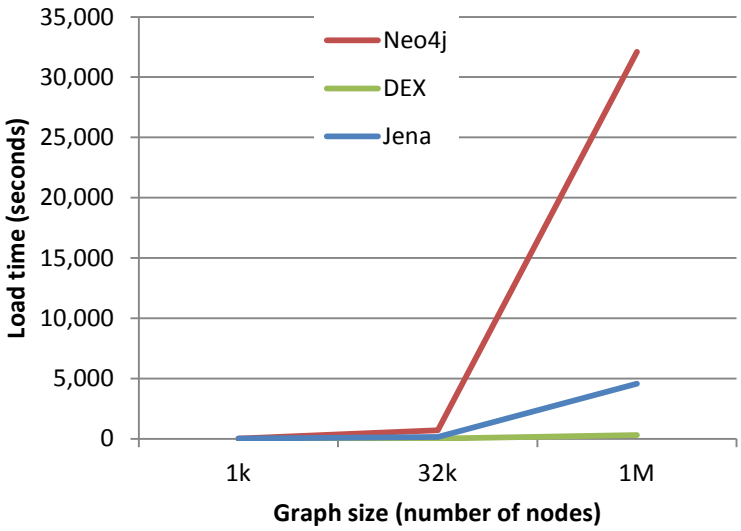

Figure 6. Load times (K1) as a function of graph size from Dominguez-Sal et al. HypergraphDB is not shown since it took 86,400 seconds to load 32k and 9.4M node graphs.

* A Lesser General Public License (LGPL) DB that does not appear to be Blueprints compliant.

† An open source Semantic Web framework that provides a Java API for loading, querying, and other manipulations of RDF graphs making it somewhat Blueprints-compliant.

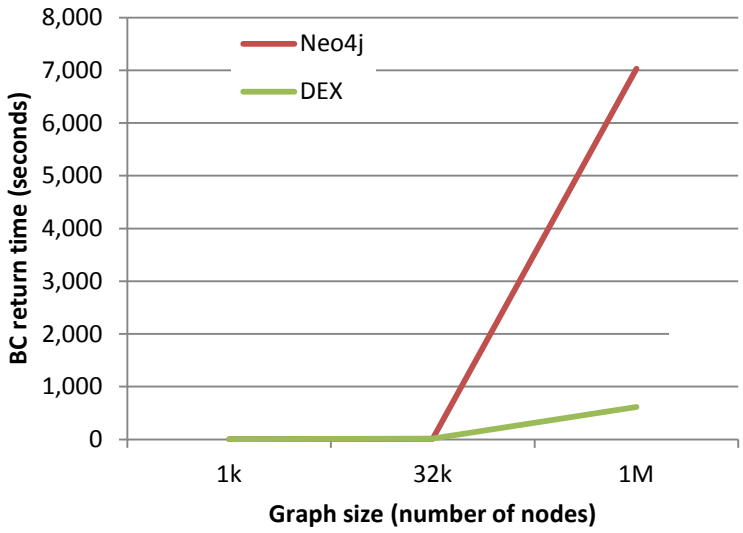

Figure 7. Betweeness centrality times (K4) as a function of graph size from Dominguez-Sal et al. Jena not shown since it took ~59,000 seconds to perform the K4 (betweeness centrality) task. HypergraphDB is not shown since load times were extremely large for 32k and 9.4M vertices graphs and therefore the computation was not applicable.

In this study, DEX is the clear favorite for the eScore requirements since it minimizes load times and has betweeness centrality time performance that scales predictably with increased graph size. Neo4j is a potential alternative.

A final study we mention is a recent performance evaluation performed by McColl and co-workers that performed an evaluation on traditional, disk-backed, ACID-compliant SQL-based relational DBs (MySQL, Oracle), new NoSQL and graph DBs (Neo4j, OrientDB, Titan), distributed graph processing technologies (Giraph, Bagel, Faunus) and in-memory graph packages (e.g. NetworkX, Gephi).[8] The authors conducted a qualitative comparison between 28 different graph DBs/processing technologies and quantitatively assessed the performance of a subset of these. The authors used the R-MAT generator to create four scale-free graphs of increasing size: 1K vertices/8K edges, 32K vertices/256K edges, 1M vertices/8M edges, and 16M vertices/256M edges. They quantitatively compared the performance across common graph analytics tasks: (1) finding single source shortest paths (SSSP),* (2) the Shiloach-Vishkin algorithm for finding connected components, (3) the Bulk Synchronous Parallel (BSP) (e.g., Pregel) power-iteration implementation of PageRank, and (4) a set of parallelized edge insertions and deletions where the updates have a 6.25% probability of being deletions. The authors used a single-node platform consisting of a system with four AMD Opteron 6282 SE processors and 256 GB of DDR3 RAM and a distributed system consisting of a 21-node cluster where each node had at least two Intel Xeon X5660 processors with 24 GB of DDR3 RAM connected by ODR Infiniband.

* All-pairs shortest paths calculations are necessary steps in many graph analytic algorithms (e.g., betweeness centrality).

Figure 8 shows the memory footprint occupied by the graph DBs/processing technology after all operations are performed on a 32k node/256k edge graph. We note that McColl et al. report that Neo4j and DEX are not able to perform the equivalent operations on a 1M node/8M edge graph. **Figure 9** shows benchmark times for nine of the graph DBs/packages they considered, performed on the 1M/8M graph.*

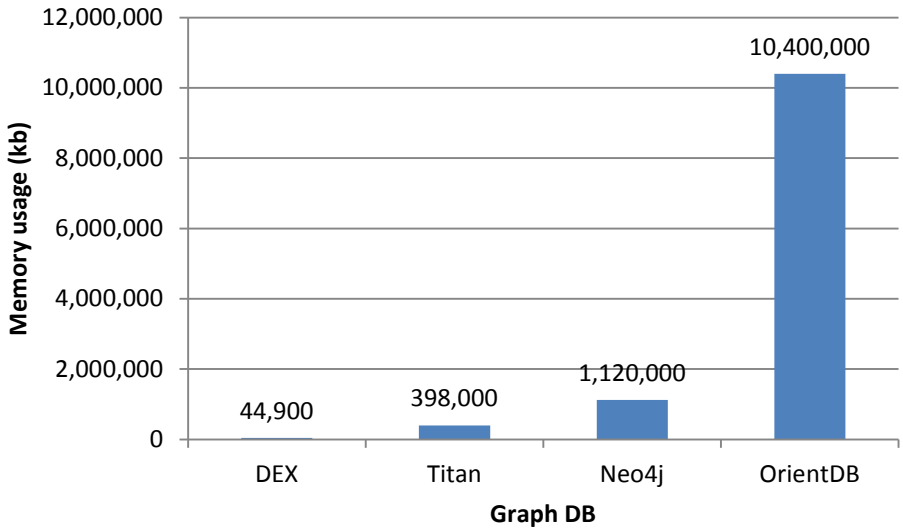

Figure 8. Memory footprint of graph DB packages after performing a series of operations on a 32k node/ 256k edge graph as reported by McColl et al. The dedicated graph DB technologies included in this figure include DEX, Titan, Neo4j, and OrientDB.

* DBs for which no results are shown in Figures 8 and 9 either ran out of memory or did not complete the tasks in a reasonable amount of time, relative to those DBs shown.

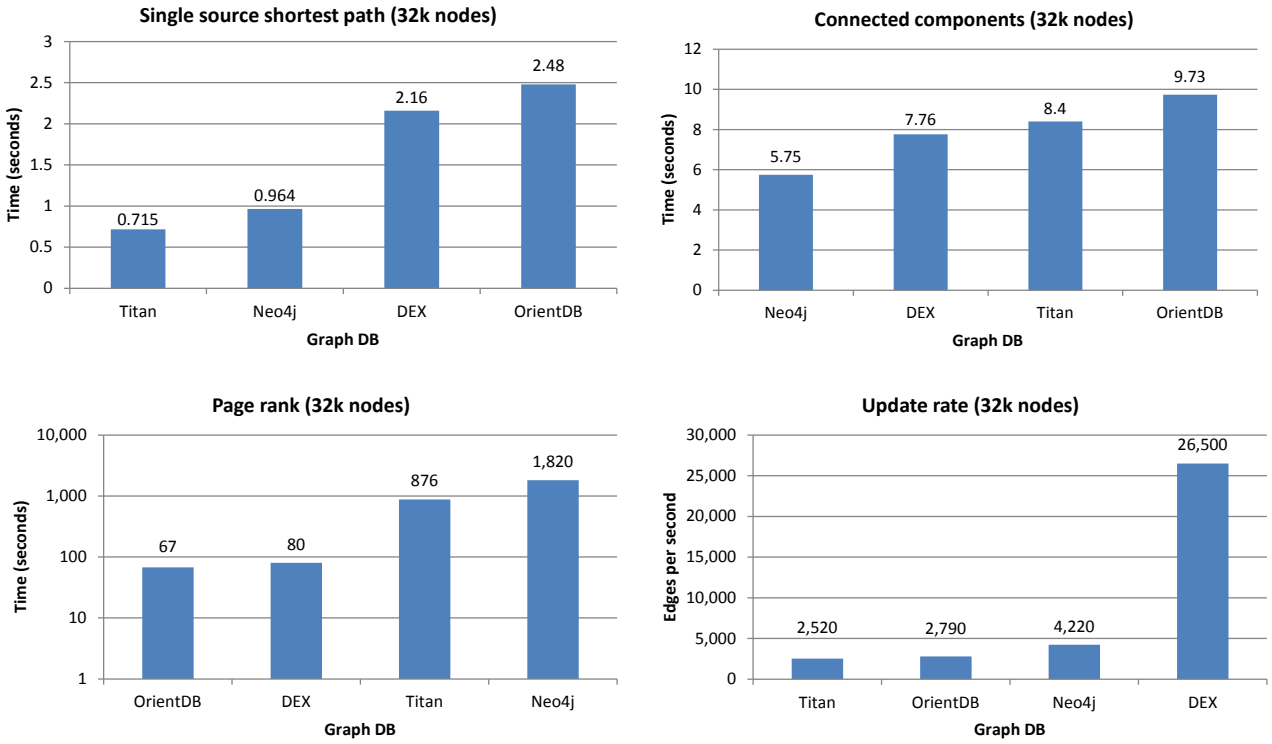

Figure 9. Benchmark performance times for graph DB packages on the four tasks for the 32k node/ 256k edge graph as reported in McColl et al.

Implications for the eScore application from this study are mixed, depending on the performance metric used. In terms of memory usage DEX is the best graph DB option, followed by Titan, then Neo4j, then OrientDB. However, when we look at various operations in Figure 9, DEX outperforms all other options in only update rate. Titan performs best in single source shortest path calculations. Neo4j performs best in connected components calculations, and OrientDB performs best with page rank calculations. McColl examined a larger 1M node/ 8M edge graph and in this graph only Titan and OrientDB were capable of completing calculations. In the larger graph, Titan outperforms OrientDB in shortest path, connected components, and update rate calculations and OrientDB outperforms Titan in page rank calculations.

Conclusions

The qualitative comparison between the different DBs shows that the different options provide a wide range of features, but DEX, Neo4j, and Titan provide distinct data feature advantages with the capability for main memory storage and attributed graphs. The quantitative comparison between different DBs confirms that DEX, Neo4j, and Titan stand out, although we might want to gather more information about Sail and SGDB which showed promise in the Ciglan study. However, we conclude that DB performance varies greatly depending on the size of the graph

and on how well-optimized a given DB is to a specific computational task. Therefore, we conclude that there is not a one-size fits all solution. It appears that DEX and Titan performance scale predictably with increased graph size and are the best alternatives across the various quantitative tests performed. Neo4j performance may be unpredictable with large graphs, but for small graphs (under 32,000 nodes) Neo4j outperforms DEX and Titan.

Based on our general eScore requirements, we conclude that the eScore approach does not leverage a specific feature offered by only one alternative. The DBs that might be best suited for eScore include DEX and Titan, but Neo4j presents a potential alternative depending upon the size of the graph.

Furthermore, we anticipate that some eScore applications will require read-write operations, particularly in instances where the graph is routinely updated. It appears that DEX and Titan outperform most alternatives in read-write workloads,[9] and therefore, these alternatives might be the best to explore further in eScore applications.

References

1. Internet site, M. Loukides, "The NoSQL Movement. How to think about choosing a database," 2012, http://strata.oreilly.com/2012/02/nosql-non-relational-database.html, accessed 22 April 2014 background
2. Working paper, M. Buerli, Department of Computer Science, Cal Poly San Luis Obispo, "The Current State of Graph Databases", December 2012, https://wiki.csc.calpoly.edu/csc560/raw-attachment/wiki/Buerli_Bibliography/, accessed 22 April 2014 background.
3. Conference proceedings, R. Angles, "A Comparison of Current Graph Database Models," 2012 IEEE 28th International Conference on Data Engineering Workshops, April 2012, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6313676, accessed 15 April 2014 background
4. Conference proceedings, R. Angles, "A Comparison of Current Graph Database Models," *2012 IEEE 28th International Conference on Data Engineering Workshops*, April 2012, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6313676, accessed 15 April 2014 background
5. Conference proceedings, S. Jouili & V. Vansteenberghe, "An empirical comparison of graph databases," *2013 ASE/IEEE International Conference on Big Data*, September 2013, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6693403, accessed 15 April 2014 background
6. Conference proceedings, M. Ciglan, A. Averbuch, & L. Hluchy, "Benchmarking traversal operations over graph databases," *2012 IEEE 28th International Conference on Data Engineering Workshops*, April 2012, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6313678, accessed 15 April 2014 background
7. Book, D. Dominguez-Sal, P. Urbon-Bayes, A. Gimenez-Vano, et al., "Survey of Graph Database Performance on the HPC Scalable Graph Analysis Benchmark," in *Web-Age Information Management*, Springer Berlin Heidelberg, 2010, Berlin, Germany, pp. 37-48 background
8. Conference proceedings, R. McColl, D. Ediger, J. Poovey, et al., "A performance evaluation of open source graph databases," *Proceedings of the first workshop on parallel processing for analytics applications*, February 2014, http://dl.acm.org/citation.cfm?id=2567638, accessed 25 April 2014 background
9. Conference proceedings, S. Jouili & V. Vansteenberghe, "An empirical comparison of graph databases," *2013 ASE/IEEE International Conference on Big Data*, September 2013, http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6693403, accessed 15 April 2014 background